



Feature Brief

Static Metrics



Cantata provides a full and unique suite of intelligent testing capabilities for the efficient unit and integration testing of C and C++. In this feature brief we highlight the ways in which the Static Metrics features of the tool can be best used.

Contents

Static Metrics.....	1
1 Cantata Static Metric Analysis Overview.....	3
1.1 PURPOSE OF CANTATA STATIC METRIC ANALYSIS.....	3
1.1.1 Code Complexity.....	3
1.1.2 Object Oriented Metrics.....	3
1.1.3 Coding Standards.....	4
1.1.4 Test Effort Planning.....	4
1.2 PERFORMING STATIC ANALYSIS WITH CANTATA.....	4
1.2.1 Detailed use of Cantata Static Analysis Metrics.....	4
1.3 THE METRICS AVAILABLE.....	5
1.3.1 Standard Code Size Metrics.....	5
1.3.2 Standard Code Quality Metrics.....	5
1.3.3 Standard Complexity Metrics.....	5
1.3.4 Specialist Object Oriented Metrics.....	6
1.3.5 Additional System Metrics.....	7
1.4 GENERATING STATIC METRICS.....	7
1.5 PROCESSING THE DATA.....	7
1.6 VISUALISING THE RESULTS.....	9

Copyright Notice

Subject to any existing rights of other parties, QA Systems GmbH is the owner of the copyright of this document. No part of this document may be copied, reproduced, stored in a retrieval system, disclosed to a third party or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of QA Systems GmbH.

© QA Systems GmbH 2016

1 Cantata Static Metric Analysis Overview

Even software which has been thoroughly dynamically tested can still have its problems. Static analysis gives the developer useful information on non-functional qualities of the source code, such as its maintainability and compliance with coding standards. Static Analysis, as the name suggests, involves the static inspection of the source code in order to provide an assessment of various non-functional features relating to the software.

1.1 Purpose of Cantata Static Metric Analysis

Cantata static analysis provides over 300 metrics on source code which can assist developers in four important areas:

- > Objective measurement of code complexity and structure
- > Object Oriented metrics assessment
- > Enforcement of coding standards
- > Test effort planning

1.1.1 Code Complexity

Cantata supports code complexity metrics as a means of increasing the maintainability of software, through objective measurement using recognised 'academic' and common sense metrics:

- > Halstead's Software Science metrics.
- > McCabe's, Myers' and Hansen's cyclomatic complexity metrics.
- > Average and maximum nesting level.
- > Basic counts of language constructs (comments, lines of code, statements, parameters etc)

1.1.2 Object Oriented Metrics

In addition to code complexity measures, Cantata also provides a number of metrics which measure aspects of object oriented implementation. These include:

- > Chidamber and Kemerer's MOOSE metric set.
- > Fernando Brito e Abreu's MOOD metric set.
- > Bansiya and Davis' QMOOD metric set.
- > Robert Martin's object-oriented dependency metrics.
- > McCabe's object-oriented metrics.
- > Bansiya's class entropy metrics.

All metrics are provided at the function, class, translation unit, or system level, as appropriate.

1.1.3 Coding Standards

Organisations are increasingly adopting coding standards as a means of improving software quality and maintainability. However, unless these standards can be verified in an automated way, it is difficult to enforce them effectively. While Cantata is not a coding standards rule checking product, it does provide the developer with static analysis metrics on the use of several useful coding constructs such as:

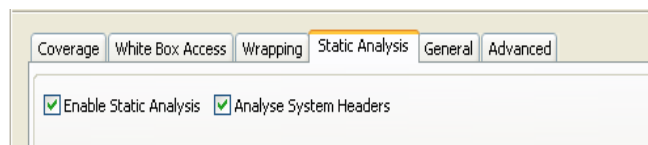
- > Unreachable code
- > Switch statements with no defaults or fallthroughs
- > Number of GOTO statements, used and unused GOTO Labels
- > Lack of cohesion methods

1.1.4 Test Effort Planning

Static analysis is often performed as part of a code review exercise, prior to dynamic testing. One important feature of Cantata static analysis is the capability to use industry standard complexity metrics to accurately estimate the testing effort for source items. An example is McCabe Cyclomatic Complexity and its variants, which the result of which strongly correlates to the minimum number of test cases required to achieve 100% decision code coverage.

1.2 Performing Static Analysis with Cantata

Cantata static analysis is invoked on a build of a Cantata enabled software project.



By default Cantata will analyze all code in the project for all 300+ static analysis metrics. Configuring the analysis can be achieved through the following methods:

- > Enabling / disabling analysis of system headers
- > Specifying which static analysis metrics to calculate (through an options file)
- > Specifying the source files, functions or classes to be analyzed
- > Specifying precisely the source code statements to be analyzed (through pragmas in the source code)

1.2.1 Detailed use of Cantata Static Analysis Metrics

Cantata Static Analysis provides hundreds of function, class and file scope code quality and complexity metrics. Code quality and complexity metrics provided by Cantata can help users to determine areas of the code that will most likely suffer from bugs, as well as producing data from which the time required for testing can be estimated. Once the metrics have been gathered by Cantata they can be processed and manipulated using an add-in for Microsoft Excel.

1.3 The Metrics Available

Some of the more common metrics provided by Cantata are described below. For an exhaustive list please refer to the Cantata Static Analysis manual.

1.3.1 Standard Code Size Metrics

These are simple metrics regarding the number of lines of code, comments, etc.

Name	Description	Scope
LINE_CODE	Total number of lines of code (including blank lines and comments).	Function or system
LINE_COMMENT	Total number of lines of comments (both C and C++).	Function or system
LINE_SOURCE	Total number of lines of source code (not including blank lines or comments).	Function or system

1.3.2 Standard Code Quality Metrics

The quality of a piece of software is to some degree based on the number of occurrences of dubious code contained within it. These metrics alert the user of such occurrences.

Name	Description	Scope
LABEL_GOTOUSED	Number of goto labels that are used.	Function or system
LABEL_GOTOUNUSED	Number of unused goto labels.	Function or system
STMT_GOTO	Number of goto statements.	Function or system
SWITCH_NODEF	Number of switch statements with no default.	Function or system
SWITCH_FALLTHRU	Number of non-empty case blocks which fall through to the next case block.	Function or system
UNREACHABLE	Number of statically unreachable statements in the given scope.	Function or system

1.3.3 Standard Complexity Metrics

The complexity of a piece of code is generally regarded as a measure that will affect the effort involved with maintaining it. These metrics attempt to estimate the complexity of the software based on various factors, such as the level of nesting.

Name	Description	Scope
HALSTEAD_PARAMS	Number of parameters.	Function
MCCABE	The McCabe Cyclomatic Complexity value for the function.	Function
NESTING_MAX	Maximum statement nesting level.	Function
NESTING_SUM	Sum of the statement nesting levels for all statements in the function.	Function

1.3.4 Specialist Object Oriented Metrics

Many standard metrics are still applicable to OO systems. For example, the maximum nesting levels within functions is also applicable to class methods. However there are also a range of specific OO metrics. These may be with respect to a given class, or for the system as a whole.

Name	Description	Scope
MAX_DEPTH	Maximum length of inheritance path to ultimate base class.	System
MOOD_AD	Number of new attributes defined for this class.	Class
MOOD_MD	Number of new methods plus overridden methods defined for this class.	Class
MOOD_AHF	Proportion of attributes that are hidden (private or protected).	Class
MOOD_MHF	Proportion of methods that are hidden (private or protected).	Class
MOOSE_CBO	Level of coupling between objects. The number of classes with which this class is coupled (via a non-inheritance dependency from this class to that, or vice versa).	System
MOOSE_WMC_MCCABE	Average McCabe Cyclomatic Complexity value of for all methods of the class (excluding inherited methods) defined in this translation unit.	Class
MOOSE_LCOM98	Chidamber & Kemerer's Lack of Cohesion of Methods metric (1998 definition). The minimum number of disjoint clusters of (new or overridden) methods (excluding constructors), where each cluster operates on disjoint set of (new) instance variables.	Class
MOOSE_RFC	Chidamber & Kemerer's Response for a class metric. The number of methods or functions defined in the class or called by methods of the class.	Class

The 'OO' aspects of the C++ language have tended to render the old procedural C metrics less useful, but fortunately new sets of metrics have taken their place. The popular ones include MOOSE (Metrics for OO Software Engineering), MOOD (Metrics for OO Design), and QMOOD (Quality Metrics for OO Design). Between them they define a number of metrics which can be useful for judging whether a C++ class is 'worth testing'. Some examples are:

Quality identified in source code	EXAMPLE METRICS
Poor or Questionable Design	'MCCABE Quality' 'MOOSE Lack of Cohesion among Methods' 'MOOD Attribute Hiding Factor'
Estimated Number of Faults	'MOOD Methods Defined' 'MOOD Attributes Defined' 'MOOSE Weighted Methods in Class'
General Complexity	'MOOSE Depth of Inheritance' 'QMOOD Number of Ancestors' 'MOOSE Number of Children'
Estimated Test Effort	'Methods Available' 'MOOSE Response for a Class' 'MOOSE Coupling Between Objects' 'MOOD Method Hiding Factor'

1.3.5 Additional System Metrics

Additional system level metrics can be created by taking averages for various class or function scope metrics. For example, we can calculate the mean McCabe Cyclomatic Complexity value for all functions or methods within our system.

1.4 Generating Static Metrics

Cantata will generate the static analysis metrics for units as they are compiled. Gathering static analysis data does not actually require the code to be compiled but this is a side effect of Cantata being called during the compile phase.

1.5 Processing the Data

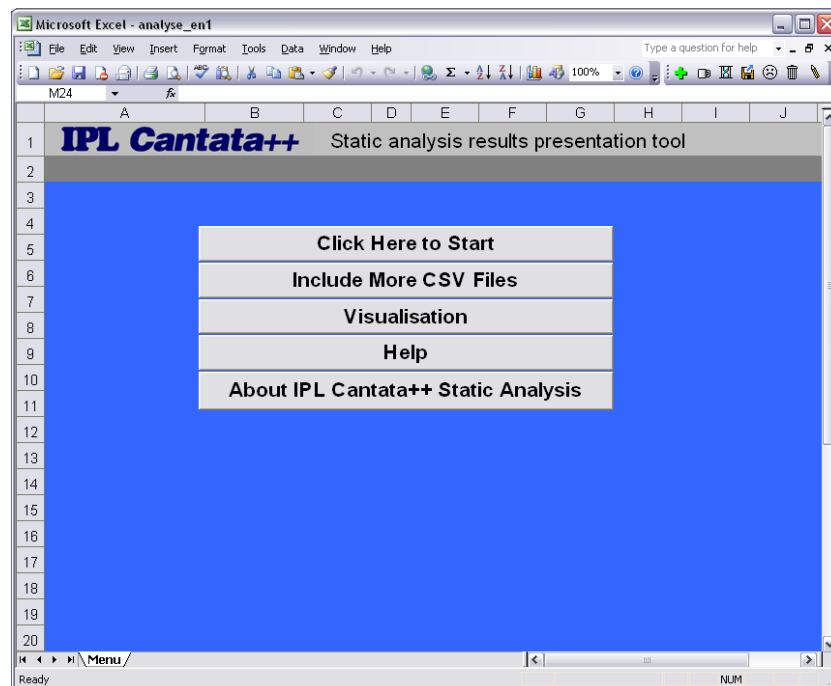
The data contained within the CSV files is raw and needs to be processed and combined to generate many of the higher-level metrics. The task of merging and combining the raw CSV data is controlled by the Cantata Static Analysis Results Presentation Tool, which is a Microsoft Excel Template spreadsheet.

If you are not running Cantata on Windows but have access to Excel on a Windows machine, the Static Analysis Results Presentation Tool can be installed separately by running the Windows installer. If you do not have access to Excel, the same functionality can be achieved using the csvmerge tool and the analyse AWK script from the bin directory.

Once loaded, you will be presented with the following screen:

Cantata Feature Brief: Static Metrics

Page | 8



Click the “Click Here to Start” button to begin. You will then be asked to select a set of CSV files for processing. You will then have to confirm that your selection is complete. Click “Yes” to continue. The Summary screen should then be displayed. This contains a summary of all the processed, combined and formatted metrics gathered for the each different scope:

	SYSTEM	CLASS	FUNCTION
1	IPL Cantata++ Static analysis results presentation tool		
2			
3	SYSTEM	CLASS	FUNCTION
4	(system)	HEADER_FILE	HEADER_FILE
5		MC_CABLE_MAXV	MC_CABLE_MAXV
6		MC_CABLE_PUBDATA	MC_CABLE_PUBDATA
7		MC_CABLE_QUAL	MC_CABLE_QUAL
8		MC_CABLE_MAX	MC_CABLE_MAX
9		MC_CABLE_CBO	MC_CABLE_CBO
10		MC_CABLE_NOC	MC_CABLE_NOC
11		MC_CABLE_NOC	MC_CABLE_NOC
12		MC_CABLE_NOC	MC_CABLE_NOC
13		MC_CABLE_NOC	MC_CABLE_NOC
14		MC_CABLE_NOC	MC_CABLE_NOC
15		MC_CABLE_NOC	MC_CABLE_NOC
16		MC_CABLE_NOC	MC_CABLE_NOC
17		MC_CABLE_NOC	MC_CABLE_NOC
18		MC_CABLE_NOC	MC_CABLE_NOC
19		MC_CABLE_NOC	MC_CABLE_NOC
20		MC_CABLE_NOC	MC_CABLE_NOC
21		MC_CABLE_NOC	MC_CABLE_NOC
22		MC_CABLE_NOC	MC_CABLE_NOC
23		MC_CABLE_NOC	MC_CABLE_NOC
24		MC_CABLE_NOC	MC_CABLE_NOC
25		MC_CABLE_NOC	MC_CABLE_NOC
26		MC_CABLE_NOC	MC_CABLE_NOC
27		MC_CABLE_NOC	MC_CABLE_NOC
28		MC_CABLE_NOC	MC_CABLE_NOC
29		MC_CABLE_NOC	MC_CABLE_NOC
30		MC_CABLE_NOC	MC_CABLE_NOC
31		MC_CABLE_NOC	MC_CABLE_NOC
32		MC_CABLE_NOC	MC_CABLE_NOC
33		MC_CABLE_NOC	MC_CABLE_NOC
34		MC_CABLE_NOC	MC_CABLE_NOC
35		MC_CABLE_NOC	MC_CABLE_NOC
36		MC_CABLE_NOC	MC_CABLE_NOC
37		MC_CABLE_NOC	MC_CABLE_NOC
38		MC_CABLE_NOC	MC_CABLE_NOC
39		MC_CABLE_NOC	MC_CABLE_NOC
40		MC_CABLE_NOC	MC_CABLE_NOC
41		MC_CABLE_NOC	MC_CABLE_NOC
42		MC_CABLE_NOC	MC_CABLE_NOC
43		MC_CABLE_NOC	MC_CABLE_NOC
44		MC_CABLE_NOC	MC_CABLE_NOC
45		MC_CABLE_NOC	MC_CABLE_NOC
46		MC_CABLE_NOC	MC_CABLE_NOC
47		MC_CABLE_NOC	MC_CABLE_NOC
48		MC_CABLE_NOC	MC_CABLE_NOC
49		MC_CABLE_NOC	MC_CABLE_NOC
50		MC_CABLE_NOC	MC_CABLE_NOC
51		MC_CABLE_NOC	MC_CABLE_NOC
52		MC_CABLE_NOC	MC_CABLE_NOC
53		MC_CABLE_NOC	MC_CABLE_NOC
54		MC_CABLE_NOC	MC_CABLE_NOC
55		MC_CABLE_NOC	MC_CABLE_NOC
56		MC_CABLE_NOC	MC_CABLE_NOC
57		MC_CABLE_NOC	MC_CABLE_NOC
58		MC_CABLE_NOC	MC_CABLE_NOC
59		MC_CABLE_NOC	MC_CABLE_NOC
60		MC_CABLE_NOC	MC_CABLE_NOC
61		MC_CABLE_NOC	MC_CABLE_NOC
62		MC_CABLE_NOC	MC_CABLE_NOC
63		MC_CABLE_NOC	MC_CABLE_NOC
64		MC_CABLE_NOC	MC_CABLE_NOC
65		MC_CABLE_NOC	MC_CABLE_NOC
66		MC_CABLE_NOC	MC_CABLE_NOC
67		MC_CABLE_NOC	MC_CABLE_NOC
68		MC_CABLE_NOC	MC_CABLE_NOC
69		MC_CABLE_NOC	MC_CABLE_NOC
70		MC_CABLE_NOC	MC_CABLE_NOC
71		MC_CABLE_NOC	MC_CABLE_NOC
72		MC_CABLE_NOC	MC_CABLE_NOC
73		MC_CABLE_NOC	MC_CABLE_NOC
74		MC_CABLE_NOC	MC_CABLE_NOC
75		MC_CABLE_NOC	MC_CABLE_NOC
76		MC_CABLE_NOC	MC_CABLE_NOC
77		MC_CABLE_NOC	MC_CABLE_NOC
78		MC_CABLE_NOC	MC_CABLE_NOC
79		MC_CABLE_NOC	MC_CABLE_NOC
80		MC_CABLE_NOC	MC_CABLE_NOC
81		MC_CABLE_NOC	MC_CABLE_NOC
82		MC_CABLE_NOC	MC_CABLE_NOC
83		MC_CABLE_NOC	MC_CABLE_NOC
84		MC_CABLE_NOC	MC_CABLE_NOC
85		MC_CABLE_NOC	MC_CABLE_NOC
86		MC_CABLE_NOC	MC_CABLE_NOC
87		MC_CABLE_NOC	MC_CABLE_NOC
88		MC_CABLE_NOC	MC_CABLE_NOC
89		MC_CABLE_NOC	MC_CABLE_NOC
90		MC_CABLE_NOC	MC_CABLE_NOC
91		MC_CABLE_NOC	MC_CABLE_NOC
92		MC_CABLE_NOC	MC_CABLE_NOC
93		MC_CABLE_NOC	MC_CABLE_NOC
94		MC_CABLE_NOC	MC_CABLE_NOC
95		MC_CABLE_NOC	MC_CABLE_NOC
96		MC_CABLE_NOC	MC_CABLE_NOC
97		MC_CABLE_NOC	MC_CABLE_NOC
98		MC_CABLE_NOC	MC_CABLE_NOC
99		MC_CABLE_NOC	MC_CABLE_NOC
100		MC_CABLE_NOC	MC_CABLE_NOC

Cantata Feature Brief: Static Metrics

Click on the other tabs to see the break down of results for that particular scope. For example, click the Function tab to see the individual values for each of the metrics with regard to each of the methods contained:

The screenshot shows a Microsoft Excel spreadsheet titled 'analyse_en1'. The spreadsheet displays static analysis results for 'IPL Cantata++'. The data is organized into columns for various metrics and rows for different functions. The 'Function' tab is selected, showing a list of functions and their corresponding metric values.

Function	Min	Max	Mean	Count	S.D.	Sum	Label_Used	Label_Got_Used	Line_Blank	Line_Code	McCabe	Nesting_Max	Stmt_Goto	Switch_Fall_Thru	Switch_Indef	Unreachable
stack::is_empty()	0	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0
stack::pop()	0	0	3	22	3	2	0	0	0	0	0	0	0	0	0	0
stack::push(int)	0	0	12	12	2	0	0	0	0	0	0	0	0	0	0	0
stack::stack()	0	0	5	5	5	5	5	5	5	5	5	5	5	5	5	5
stack::stack(const &)	0	0	13	8	12	1	0	0	0	0	0	0	0	0	0	0
	0	0	6	60	10	4	0	0	0	0	0	0	0	0	0	0

The buttons at the base of each metric label can be used to filter the rows that are displayed.

1.6 Visualising the Results

Although the formatted textual static analysis results are very useful it is often more applicable to visualise the data within a graph. Plotting graphs of the data can aid understanding and create overall pictures of the trends occurring lower down that are not immediately obvious when you are reading the textual data.

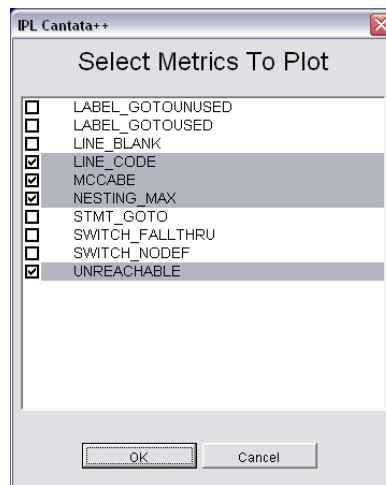
Graphs are easy to create in Microsoft Excel but first we need to create a table of the data we would like to plot. Return to the Menu tab of the spreadsheet and click the “Visualisation” button. This will display a dialog offering scopes for the data that you can use. Generally these will include:

- > Class
- > Function
- > Category

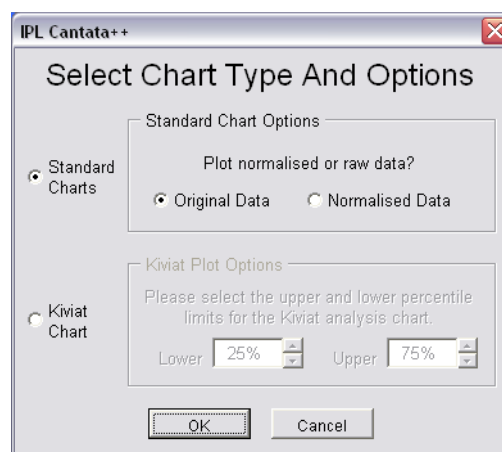
To begin with, select the “Function” option.

Now you must select the metrics to plot. You should select a set of procedural style metrics that will evaluate code size, quality and complexity for all of the methods/functions in the code:

Cantata Feature Brief: Static Metrics



Click “OK” to continue. You will now have the option to select the chart type. The “Standard Charts” are all of the chart types available within Excel, which is a considerable set. The “Kiviart Chart” is specific to the Cantata Template and resembles a radial graph:



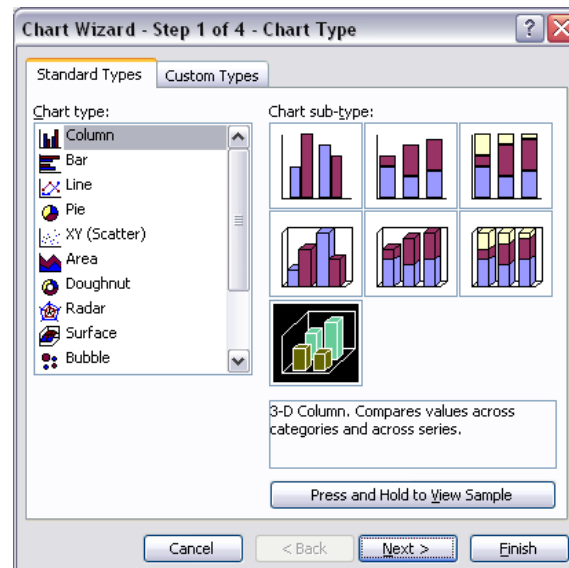
Click “OK” to continue.

If you elected to use one of the standard charts then a table of the specific data will be produced:

	A	B	C	D	E	F	G	H
1	Function	UNREACHABLE	NESTING_MAX	MCCABE	LINE_CODE			
2	stack.is_empty()	0	0	1	4			
3	stack.pop()	0	1	2	12			
4	stack.push(int)	0	2	3	22			
5	stack.stack()	0	0	1	4			
6	stack.stack(const &)	0	1	3	18			
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								

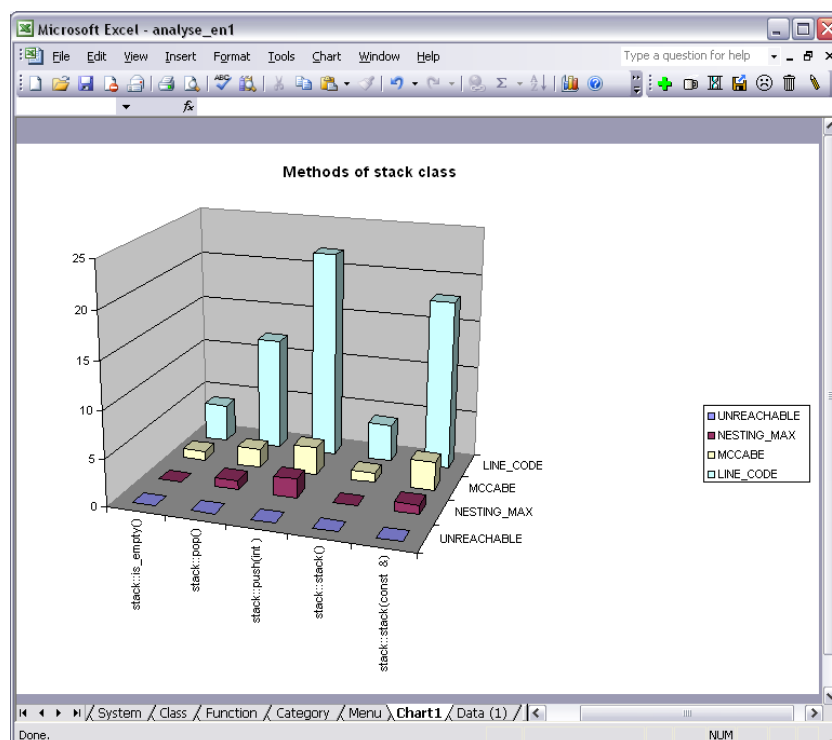
Cantata Feature Brief: Static Metrics

Click the chart icon to start creating the graph:



Select the 3D Column graph and make sure that the data table is in an order where the bars will generally get higher toward the rear of the graph.

This will give you something that looks like this:

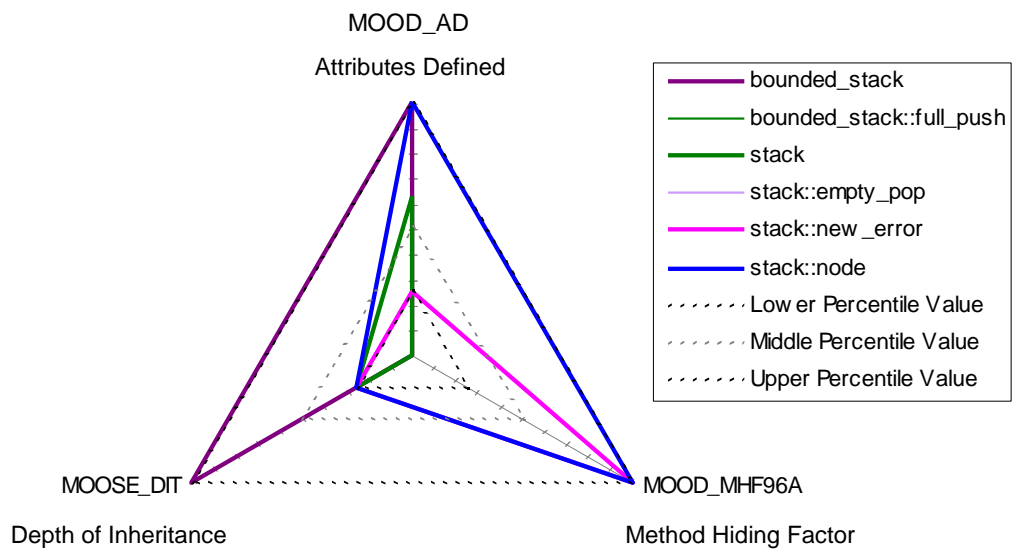


The diagrams below were produced, using the Cantata Static Analysis Presentation Tool.

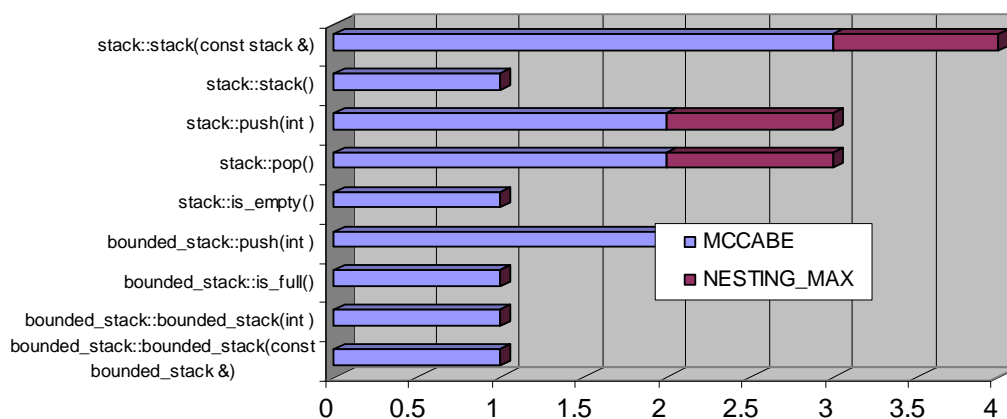
Cantata Feature Brief: Static Metrics

Page | 12

Class Kiviat plot for 2 classes



Complexity graph for 2 classes



Code and Commenting Standards Graphic

